Sequencer: smart control of components

Dr. Pierre Vignéras
pierre.vigneras@bull.net
Plan

- Overview
  - Customer Needs : EPO
  - Problems
  - Other requirements
  - Architecture
  - Incremental Use versus Black Box Use

- Details
  - DGM Algorithm
  - ISM Algorithms Overview
  - ISE Overview

- Conclusion
  - Results on the Tera-100
  - Comparison with other products
  - Summary and Future Works
Customer Request

- Emergency Power Off (EPO) of the Tera-100 (#9th in the TOP500 list)
  - > 4000 bullx Serie S servers (alias 'MESCA')
  - More than a hundreds of cold doors (Bull water-based cooling system)
  - Dozens of disk arrays (DDN SFA10K)

- Hardware should be preserved
  - Do not poweroff a cold door if at least one node is running inside the related rack

- Filesystems should be preserved (Lustre)
  - Hard power off forbidden!

- In less than 30 minutes
  - Average time for powering off (softly) a node : ~60 seconds.
Problems

- Cluster = set of heterogeneous devices
- Start/Stop : a complex task
  - Many commands
    - Nodes: ipmitool
    - Disk Array: specific to manufacturer (EMC, DDN, LSI, ...)
    - Daemon (e.g : Lustre): shine (if no HA otherwise it might be different)
  - Order should be respected
    - Stop devices cooled by a Bull cold door before the cold door itself except for the connecting switch
    - Stop io nodes before their connected disk array controllers
- Scalability:
  - independant stuff should be done in parallel where possible
- Handling failures correctly
  - E.g : a node cannot be stopped -> do not stop the related cold door
Customer Needs

- **Maximum Configurability**
  - Dependency between components and component types
  - Rules for fetching dependencies of a given component (*depsFinder*)
  - Actions to be executed on the component (not only start/stop)

- **Poweron/Poweroff**
  - Of a hardware or software component set (e.g.: rack, lustre servers)
  - Of a unique component (cold door, switch, NFS server) taking dependency into account (or not)
  - Verification and modification before actual execution

- A poweron/poweroff instruction sequence should be validated before pushing to production
Three stages:

- **Dependency Graph Maker (DGM)**
  - From dependency rules defined in a database
  - From components given in input
    - E.g: input == cold door -> poweroff all cooled nodes before

- **Instruction Sequence Maker (ISM)**
  - Find an instruction sequence that conforms to constraints expressed in the dependency graph given in input
  - Allow parallelism to be expressed in the output instruction sequence

- **Instruction Sequence Executor (ISE)**
  - Execute the instruction sequence given in input
    - Make use of parallelism where possible
    - Handle failures
BlackBox mode

- Components List
- Dependency Rules
- Sequencer
- Execution

Example: sequencer softstop colldoor[1-3] rack[4-5] compute[100-200]
Incremental mode

At each step, it is possible to check and to modify the output of the previous step and the input of the next step.

It is possible to write an input step « by hands ».

Components List
Dependency Rules
DGM
Dependency Graph
Check/Modify
ISM

Execution
ISE
Check/Modify
Instructions Sequence
BlackBox Mode vs Incremental Mode

- **BlackBox mode**
  - for simple non-critical task
    - Power off a small set of nodes
    - Power on a whole rack
  - Simple to use

- **Incremental Mode**
  - For critical task requiring validation
    - Emergency Power off the whole cluster
    - Power on the whole cluster
  1) Generate the script (DGM + ISM)
  2) Adapt the script to your needs
  3) Test the script
  4) Push the script to production
Details

- Sequencer Table
- DGM Algorithm
- ISM Algorithms Overview
- ISE Overview
Sequencer Table

- One table for all dependency rules
  - Grouped into a set called 'ruleset' (e.g: start, stop, stopForce)
  - One line in this table = one dependency rule

- Columns:
  - RuleSet : ruleset the rule is a member of
  - SymbolicName : unique name of the rule
  - ComponentType : the component type this rule applies to
  - Filter : the rule applies only to components that are filtered in
  - Action : the action to execute on the component
  - DepsFinder : tells which components a given component depends on
  - DependsOn : tells which rule should be applied to component returned by the 'deptsfinder'
  - Comments : free comments
## Sequencer Table: Example

<table>
<thead>
<tr>
<th>Rule Set</th>
<th>Symbolic Name</th>
<th>Component Type</th>
<th>Filter</th>
<th>Action</th>
<th>DepsFinder</th>
<th>DependsOn</th>
<th>Comments</th>
</tr>
</thead>
</table>
| stop     | coldoorOff    | coldoor@hw     | ALL     | bsmpower -a off  
%component | find_coldoorOff  
_dep  
%component | nodeOff | PowerOff nodes before a cold door |
| stop     | nodeOff       | compute@node   | ALL     | nodectl poweroff  
%component | find_nodeoff_deps  
%component | nfsDown | Unmount cleanly and shutdown nfs properly before halting. |
| stop     | nfsDown       | nfsd@soft      | ALL     | @/etc/init.d/nfs stop  
%component | find_nfs_client  
%component | umountNFS | Stopping NFS daemons: take care of clients! |
| stop     | umountNFS     | umountNFS@soft | ALL     | Echo WARNING: NFS mounted! | NONE | NONE | Print a warning message for each client |
| start    | coldoorStart  | coldoor@hw     | ALL     | bsmpower -a on  
%component | NONE | NONE | No dependencies |
| start    | nodeOn        | compute@node   | %name =~ compu te12 | nodectl poweron  
%component | find_nodeon_deps | coldoorStart | Power on cold door before nodes. |
| stopF orce| daOffForce    | da@hw          | %name !~ .* | da_admin poweroff  
%component | find_daOff_deps | ioServerDown | Unused thanks to Filter |

...
Sequencer Table : rules graph

Rules graph = graphical representation for a given ruleset
E.g : sequencer graphrules stop

Usefull to grasp the overall picture of a given ruleset.
Details

- Sequencer Table
- DGM Algorithm
- ISM Algorithms Overview
- ISE Overview
DGM Algorithm : Use Case

- **Input**: Ruleset='stop' & Components=(nfs1#nfsd@soft, cd0@hw, nfs2@node)

- **Purpose**:
  - stop nfsd of 'nfs1' node,
  - poweroff cold door 'cd0' and node 'nfs2'.

- **Hypothesis**:
  - nfs1 is an NFS server in a rack cooled by 'cd0', it is also an 'nfs2' client;
  - nfs2 is an NFS server not cooled by 'cd0', it is also an 'nfs1' client;
  - c1 is a compute node which is both an 'nfs1' and 'nfs2' client.

- **Constraints**:
  - Poweroff c1 before 'cd0';
  - Stop NFS daemons on 'nfs1' and 'nfs2' cleanly
  - Print a warning for each NFS client
  - Stop nfs2 cleanly
DGM Algorithm

- Initial creation of dependency graph (from input list)
  - A node in this graph has the form: (component, type)
    - nfs1#nfssd@soft
    - cd0#coldoor@hw
    - nfs2#nfs@node

- Choosing a component for rules application
  - First component matching a root rule in the graph rules
    • 'coldoorOff' is the only root and 'cd0' matches.
    • If no component matches, remove roots from the graph rules (virtually), and start again with the resulting graph rules.

- For the chosen component:
  - The depsfinder is called: it returns a node list (c,t) that should be inserted in the dependency graph
DGM Algorithm

The depsfinder of cd0 returns c1#compute and nfs1#nfs. They are both added to the graph. c1#compute is processed. Its depsfinder does not return anything. The action for its related rule is registered.

- nfs1#nfsd@soft
- cd0#coldoor@hw
- nfs2#nfs@node

- nodeOff
  - nfs1#nfs@node
  - c1#compute@node
    - [nodectrl poweroff c1]
Then, nfs1#nfs is processed. Its depsfinder returns nfs1#nfsd. This node is already in the graph. Therefore, only the link between nfs1#nfs and nfs1#nfsd is made.
This node is then processed. New dependencies are: 'c1#unmountNFS@soft' and 'nfs2#unmountNFS@soft'. These nodes match rule 'umountNFS'. They have no dependency. Their actions are recorded. Then, node nfs1#nfsd@soft is updated and finally nfs1#nfs@node.
Finally, moving up in the stack, it remains cold door action to be added on 'cd0'

Remaining in the input components list: 'nfs1#nfsd@soft' and 'nfs2#nfs@node'.

nfs1#nfsd@soft has already been processed.

We search, in the rules graph, the first component which match a root rule.
DGM Algorithm

Remaining non-processed components in the input: 'nfs2#nfs@node'

We search, in the rules graph, the first component which match a root rule.

There is none.
DGM Algorithm

Remaining non-processed components in the input: 'nfs2#nfs@node'

We thus remove (virtually) from the rules graph all roots, resulting in a new rules graph.

In this new graph, 'nfs2#nfs@node' matches 'nodeOff' rule. It is therefore the starting element for the application of next dependency rules.
DGM Algorithm

Current dependency graph:

nfs1#nfsd@soft
[ssh nfs1 /etc/init.d/nfs stop]

nfs1#nfs@node
[nodedetrl poweroff nfs1]

cd0#coldoor@hw
[bsm_power -a off_force cd0]

nfs2#nfs@node

nfs2#unmountNFS@soft
[WARNING : nfs mounted!]

c1#compute@node
[nodedetrl poweroff c1]

nfs1#unmountNFS@soft
[WARNING : nfs mounted!]

c1#unmountNFS@soft
[WARNING : nfs mounted!]
DGM Algorithm

The depsfinder applied to 'nfs2#nfs' returns 'nfs2#nfsd@soft'.
The dependency graph is updated.

```
nfs1#nfsd@soft [ssh nfs1 /etc/init.d/nfs stop]
```

```
ncd0#coldoor@hw [bsm_power -a off_force cd0]
```

```
nfs1#nfs@node [nodedet! poweroff nfs1]
```

```
c1#compute@node [nodedet! poweroff c1]
```

```
nfs2#nfs@node
```

```
nfs2#nfsd@soft
```

```
c1#unmountNFS@soft [WARNING : nfs mounted!]
```

```
nfs2#unmountNFS@soft [WARNING : nfs mounted!]
```
DGM Algorithm

The related depsfinder returns 'c1#unmountNFS@soft' and 'nfs1#unmountNFS@soft'.
'c1#unmountNFS@soft is already in the dependency graph. These nodes do not have any dependency. Action on new node is updated.

nfs1#nfsd@soft
[ssh nfs1 /etc/init.d/nfs stop]

nfs1#nfs@node
[nodectrl poweroff nfs1]

nfs2#nfs@node
[nodectrl poweroff c1]

nfs2#nfsd@soft

nfs1#nfsd@soft

cd0#coldoor@hw
[bsm_power -a off_force cd0]

nfs1#nfs@node

nfs2#unmountNFS@soft
[WARNING : nfs mounted!]

nfs1#unmountNFS@soft
[WARNING : nfs mounted!]

nfs1#unmountNFS@soft
[WARNING : nfs mounted!]

nfs2#nfsd@soft

nfs1#nfsd@soft
Finally, moving up in the stack, actions are updated.
DGM Algorithm

Final Dependency Graph:

- **cd0#coldoor@hw**
  - [bsm_power -a off_force cd0]

- **nfs1#nfs@node**
  - [nodectrl poweroff nfs1]

- **nfs2#nfs@node**
  - [nodectrl poweroff nfs2]

- **nfs1#nfsd@soft**
  - [ssh nfs1 /etc/init.d/nfs stop]

- **nfs2#unmountNFS@soft**
  - [WARNING: nfs mounted!]

- **c1#compute@node**
  - [nodectrl poweroff c1]

- **nfs1#unmountNFS@soft**
  - [WARNING: nfs mounted!]

- **nfs2#nfsd@soft**
  - [ssh nfs2 /etc/init.d/nfs stop]

- **c1#unmountNFS@soft**
  - [WARNING: nfs mounted!]
Details

- Sequencer Table
- DGM Algorithm
- ISM Algorithms Overview
- ISE Overview
Instructions Sequence Maker (ISM)

- sequencer seqmake [-f|--file file]
- Input from a file containing a dependency graph
  - Computed by previous stage (DGM)
  - Edited by hands
- XML File Format
  - Taken from the open-source python-graph library (Google)
  - Checking and Modifications by hands possible
- Output a computed instructions sequence
  - Conform to constraints expressed in the dependency graph
  - Simple: only 3 kind of « instructions »
    - 'Action': the actual command that should be executed
      - Various attributes in particular: Deps=explicit dependencies,
    - 'Seq': a sequence of instructions (implicit dependencies)
    - 'Par': independent instructions that might be executed in parallel
<seq>
  <par>
    <action component_set="c1#unmountNFS@soft" id="1">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="nfs1#unmountNFS@soft" id="2">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="nfs2#unmountNFS@soft" id="3">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="c1#compute@node" id="4">
      nodectrl poweroff c1;
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfsd@soft" id="5">
      ssh nfs1 /etc/init.d/nfs stop
    </action>
    <action component_set="nfs2#nfsd@soft" id="6">
      ssh nfs2 /etc/init.d/nfs stop
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfs@node" id="7">
      nodectrl poweroff nfs1
    </action>
    <action component_set="nfs2#nfs@node" id="8">
      nodectrl poweroff nfs2
    </action>
  </par>
  <par>
    <action component_set="cd0#coldoor@hw" id="9">
      nodectrl poweroff cd0
    </action>
  </par>
</seq>
Express a sequence

```xml
<seq>
  <par>
    <action component_set="c1#unmountNFS@soft" id="1">
      echo "WARNING : nfs mounted"
    </action>
    <action component_set="nfs1#unmountNFS@soft" id="2">
      echo "WARNING : nfs mounted"
    </action>
    <action component_set="nfs2#unmountNFS@soft" id="3">
      echo "WARNING : nfs mounted"
    </action>
    <action component_set="c1#compute@node" id="4">
      nodectrl poweroff c1
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfsd@soft" id="5">
      ssh nfs1 /etc/init.d/nfs stop
    </action>
    <action component_set="nfs2#nfsd@soft" id="6">
      ssh nfs2 /etc/init.d/nfs stop
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfs@node" id="7">
      nodectrl poweroff nfs1
    </action>
    <action component_set="nfs2#nfs@node" id="8">
      nodectrl poweroff nfs2
    </action>
  </par>
  <par>
    <action component_set="cd0#coldoor@hw" id="9">
      nodectrl poweroff cd0
    </action>
  </par>
</seq>
```
ISE Input File Format Example

```xml
<seq>
  <par>
    <action component_set="c1#unmountNFS@soft" id="1">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="nfs1#unmountNFS@soft" id="2">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="nfs2#unmountNFS@soft" id="3">
      echo "WARNING : nfs mounted";
    </action>
    <action component_set="c1#compute@node" id="4">
      nodectrl poweroff c1;
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfsd@soft" id="5">
      ssh nfs1 /etc/init.d/nfs stop
    </action>
    <action component_set="nfs2#nfsd@soft" id="6">
      ssh nfs2 /etc/init.d/nfs stop
    </action>
  </par>
  <par>
    <action component_set="nfs1#nfs@node" id="7">
      nodectrl poweroff nfs1
    </action>
    <action component_set="nfs2#nfs@node" id="8">
      nodectrl poweroff nfs2
    </action>
  </par>
  <action component_set="cd0#coldoor@hw" id="9">
    nodectrl poweroff cd0
  </action>
</seq>
```

Express parallelism
Several algorithms

- 'seq' = Topological sort
  - Trivial sequence (uses 'seq' and 'action' only)
  - Pros: high readability
  - Cons: not scalable since it only allows sequential execution

- 'par' = parallel
  - Trivial parallel (uses 'par' and 'action' only)
  - Pros: highest scalability
  - Cons: not readable by a human

- 'mixed' = level by level
  - Encapsulates leaf nodes into a 'par' instruction, remove them, start again.
  - Encapsulates all such 'par' into a 'seq'
  - Pros: readability, better performance than 'seq'
  - Cons: may produce huge graph, performance not equivalent to 'par', ...

- 'optimal'
  - Pros: performance equivalent to 'par', readability equivalent to 'mixed'
  - Cons: time to compute
ISM Algorithm Examples

On our example:

- **cd0#coldoor@hw**
  
  `[bsm_power -a off_force cd0]`

- **nfs1#nfs@node**
  
  `[nodectrl poweroff nfs1]`

- **c1#compute@node**
  
  `[nodectrl poweroff c1]`

- **nfs1#nfsd@soft**
  
  `[ssh nfs1 /etc/init.d/nfs stop]`

- **nfs2#unmountNFS@soft**
  
  `[WARNING : nfs mounted!]`

- **nfs1#unmountNFS@soft**
  
  `[WARNING : nfs mounted!]`

- **nfs2#nfs@node**
  
  `[nodectrl poweroff nfs2]`

- **nfs2#nfsd@soft**
  
  `[ssh nfs2 /etc/init.d/nfs stop]`

- **nfs2#unmountNFS@soft**
  
  `[WARNING : nfs mounted!]`

- **c1#unmountNFS@soft**
  
  `[WARNING : nfs mounted!]`
ISM Algorithm Examples

'seq' algorithm : 9 steps

1. c1#unmountNFS@soft
   [WARNING : nfs mounted!]

2. nfs2#unmountNFS@soft
   [WARNING : nfs mounted!]

3. nfs1#unmountNFS@soft
   [WARNING : nfs mounted!]

4. nfs1#nfsd@soft
   [ssh nfs1 /etc/init.d/nfs]

5. nfs1#nfs@node
   [nodectrl poweroff]

6. c1#compute@node
   [nodectrl poweroff c1]

7. cd0#coldoor@hv
   [bsm_power -a off_force]

8. nfs2#nfsd@soft
   [ssh nfs2 /etc/init.d/nfs]

9. nfs2#nfs@node
   [nodectrl poweroff nfs2]
ISM Algorithm Examples

'mixed' algorithm : 4 steps

1. c1#unmountNFS@soft [WARNING : nfs mounted!]
2. c1#compute@node [nodectrl poweroff c1]
3. nfs1#nfs@node [nodectrl poweroff nfs1]
4. cd0#coldoor@hw [bsm_power -a off_force]

1. nfs1#nfsd@soft [ssh nfs1 /etc/init.d/nfs stop]
2. nfs2#nfsd@soft [ssh nfs2 /etc/init.d/nfs stop]
3. nfs2#unmountNFS@soft [WARNING : nfs mounted!]
4. nfs1#unmountNFS@soft [WARNING : nfs mounted!]

pierre.vigneras@bull.net
Details

- Sequencer Table
- DGM Algorithm
- ISM Algorithms Overview
- ISE Overview
Instructions Sequence Executor

- sequencer seqexec [-F|--Force] [-f|--file file]
- Input from a file containing:
  - An instructions list
    • Computed by previous stage (ISM)
    • Generated by hands
- Output:
  - All messages displayed by actions
    • Prefixed by the action id for usability
  - Various reports
  - Statistics
Instructions Sequence Executor

- ISE uses ClusterShell as the back-end execution engine
  - ClusterShell is open-source (developed by CEA, used on Tera-100)
  - ClusterShell is designed for scalability

- Parallel Instructions might be executed in parallel
  - 'fanout' option defines how many actions can be launched in parallel.
    - Fanout=1 -> sequential !
    - Fanout = 1000 -> huge load on the host should be expected !

- Smart Handling of Failures
  - Returned code for actions : OK, WARNING=75, KO (=anything else)
    - Option –Force -> WARNING ~ OK, otherwise, WARNING ~ KO
  - When an action is ~ KO reverse dependencies (parents in the dependency graph) are not executed at all !
    - Prevent a cold door from being powered off if a node has not been powered off normally.
Conclusion

- Result on the Tera-100
- Comparison with other products
- Future Works
Result on the Tera-100

- **Power Off**: DGM=2m1s, ISM=6.3s [4606 actions]
  - Original Request : < 30 minutes
  - Result (2011-09): 9 minutes and 23 seconds
    - 97.7% of actions executed (successfully or not)
    - 15.3% of actions that ends on error for various reasons
    - 3.3% of actions not executed (because at least 1 dep. went on error)
  - Parameters : fanout = 500

- **Power On**: DGM=13m40s, ISM=4.9s [4604 actions]
  - Original Request : < 60 minutes
  - Result (2011-06): 4 minutes and 27 seconds
    - 99.7% of actions executed (successfully or not)
    - 6.6% of actions that ends on error for various reasons
    - 0.3% of actions not executed (because at least 1 dep. went on error)
  - Parameters : fanout = 1000 (!)
Comparison with other products

- **PowerOff/PowerOn standard solutions**
  - Sun/Oracle cluster shutdown, IBM xcat : customization ?

- **Dependency graph makers**
  - Make, scons, ant,... : focus is on files, not on hosts
  - Init, smf, launchd, upstart, systemd : no input parameters

- **Command Dispatchers**
  - Fabric, Func, Capistrano : dependencies ?

- **Workflows systems**
  - YAWL, Bonita, Intalio, jBPM, Activiti : user task oriented (too heavy)
  - ControlTier/RunDeck :
    - Main similarities : execution of workflows, failure handling
    - Main differences :
      - targets applications rather than hardwares
      - Runs as daemons (sequencer is a simple command)
      - Java-centric (JDBC, JAAS, Servlet, ...) ;-(

pierre.vigneras@bull.net
Summary and Future Works

- **Summary**
  - New original product for controlling hardware and softwares
  - Predictive: through incremental mode or reporting
  - Easy: through blackbox mode or the ISE
  - Fast: Tera-100 > 4000 nodes: poweron ~ 5 mn, poweroff ~ 10 mn
  - Smart: respect expressed constraints
  - Robust: failures are taken into account for each component

- **Future Works includes**
  - Smarter failure handling
  - Live reporting/monitoring
  - Performance improvement of dependency graph generation
  - Post-mortem reporting
  - Replaying

Dr. Pierre Vignéras
pierre.vigneras@bull.net